Nekbone

Summary Version
2.1.2

Purpose of Benchmark
Nekbone captures the basic structure and user interface of the extensive Nek5000 software which is a high order, incompressible Navier-Stokes solver based on the spectral element method. Nekbone solves a standard Poisson equation using a conjugate gradient iteration with a simple preconditioner on a block or linear geometry. Nekbone exposes the principal computational kernel to reveal the essential elements of the algorithmic-architectural coupling that is pertinent to Nek5000. More details on the benchmark are provided in the readme.pdf file included in the distribution.

Characteristics of Benchmark
Nekbone solves a standard Poisson equation using the spectral element method with an iterative conjugate gradient solver with a simple preconditioner. The computational domain is partitioned into high-order quadrilateral elements. Based on the number of elements, number of processors, and the parameters of a test run, Nekbone allows a decomposition that is either a 1-dimensional array of 3D elements, or a 3-dimensional box of 3D elements. The benchmark is highly scalable and can accommodate a wide range of problem sizes, specified by setting the number of spectral elements and the polynomial order of the elements by editing the appropriate build and input files. The benchmark consists of a setup phase and a solution phase. The solution phase consists of conjugate gradient iterations that call the main computational kernel, which performs a matrix vector multiplication operation in an element-by-element fashion. Overall each iteration consists of vector operations, matrix-matrix multiply operations, nearest-neighbor communication, and MPI Allreduce operations. The benchmark is written in Fortran and C, where C routines are used for the nearest neighbor communication and the rest of the compute kernel routines are in Fortran. The current version of the benchmark uses MPI parallelism with no threading.

Mechanics of Building Benchmark:
- Change to the nekbone *test/example1* directory
- Edit the *SIZE* file, if needed, to:
  - set the maximum number of MPI ranks, *lp*
  - set the maximum number of elements per rank, *lelt*
- Edit the *makenek* script and set:
  - SOURCE_ROOT to the path to the nekbone source code
  - F77 to the name of the Fortran compiler
  - CC to the name of the C compiler
  - Uncomment IFMPI flag if not using MPI

    - Uncomment and specify any link flags in USR_LFLAGS
    - Uncomment and specify optimization flags, OPT_FLAGS_STD and OPT_FLAGS_MAG
- Run the *makenek* script. If *makenek* fails to recognize the compiler it will generate a *makefile* and stop, the *makefile* may then be edited manually and the code compiled by running *make.*
- Run '*make clean*' to remove previous build

Mechanics of Running Benchmark

1. Small problem: single node and/or single CPU
    - Compile as described above with or without MPI
    - run *nekbone*
2. Medium problem: (<1K node) job
    - Compile as described above with MPI enabled
    - run *nekbone*
3. Large Sequoia problem:
    - The Large Sequoia problem is defined as having 50,331,648 spectral elements. The values in the data.rea file that determine the polynomial orders (nx0, nxN, and nxD) are to be set to 9, 12, and 3 respectively. On Sequoia the problem was run using 6,291,456 ranks in a 384x128x128 distribution with 8 elements per rank in a 2x2x2 distribution. The number of MPI ranks and elements per rank used maybe any value so long as total spectral element count remains 50,331,648 elements. The values controlling the polynomial orders must remain fixed at 9,12, and 3.
    - Compile as described above with MPI enabled and altering *lp* and *letl* in SIZE as required
    - Edit data.rea to set the appropriate number of elements per process.
    - run nekbone
4. CORAL class problem:
    - The CORAL problem is defined as having approximately 2.5 million spectral elements per petaflop of theoretical peak performance of the system. The number of elements used may be any value within 5% of the approximate value calculated for the system. A 100 PF system therefore should use a total element count within 5% of 250 million elements.
    - The total element count is the number of MPI ranks multiplied by the number of elements per rank. The number of MPI ranks and number of elements per rank must be chosen such that the total element count meets the criteria specified above.
    - The number of elements per rank is specified in the data.rea file by setting values for iel0, ielN, and ielD. The values set for iel0 and ielN should be the same and ielD should remain set to 1.
    - The number of MPI ranks must be chosen such that the 3D process distribution used has process counts in the X, Y, and Z dimensions (reported as npx, npy, and npz in the standard output) that are all greater

than 2 and the ratio of the largest to smallest value in the set (npx, npy, npz) is not greater than 5, unless restricted by unresolvable hardware limitations. In cases where the value of this ratio exceeds 5 it must be shown that no lower ratio could be achieved. For example (npx = 20, npy = 15, npz = 7) requires no justification, while (npx = 79, npy = 17, npz = 13) should be justified by hardware limitations.

- The process distribution (npx, npy, npz) may be set manually, provided it meets the criterion above, by specifying npx, npy, and npz values in the data.rea input. The benchmark will generate a distribution with a minimal ratio if the value of npx*npy*npz does not equal the number of MPI ranks.
- The local element distribution (mx, my, mz) may be set manually to any value by specifying mx, my, and mz values in the data.rea input. The benchmark will generate a distribution with a minimal ratio if the value of mx*my*mz does not equal the value set for the local element count.
- The values in the data.rea file that determine the polynomial orders (nx0, nxN, and nxD) are to be set to 9, 12, and 3 respectively.
- Compile as described above with MPI enabled and altering *lp* and *letl* in SIZE as required
- run nekbone

Verification of Results

The FOM to be reported is the average aggregate MFlop rate calculated and reported in the nekbone output as “Av MFlops”. Benchmark results are considered correct if the reported rnorm is small, generally less than $1x10^{-8}$, after 101 conjugate gradient iterations.